Jupyter: Thinking and Storytelling with Code and Data

Brian Granger[1,2] and Fernando Pérez[3,4]

[1]Amazon Web Services
[2]California Polytechnic State University, San Luis Obispo
[3]Department of Statistics, UC Berkeley

[4]Department of Data Science and Technology, Computational Research Division, Lawrence Berkeley National Laboratory

February 11, 2021

Abstract

Project Jupyter is an open-source project for interactive computing widely used in data science, machine learning, and scientific computing. We argue that even though Jupyter helps users perform complex, technical work, Jupyter itself solves problems that are fundamentally human in nature. Namely, Jupyter helps humans to think and tell stories with code and data. We illustrate this by describing three dimensions of Jupyter: interactive computing, computational narratives, and the idea that Jupyter is more than software. We illustrate the impact of these dimensions on a community of practice in Earth and climate science.

Project Jupyter[1] is an open-source software project and community that builds software, services, and open standards for interactive computing across dozens of programming languages. The core of Jupyter is the Jupyter Notebook [1], an open document format and web application that enables users to compose and share interactive programs that combine live code with narrative text, equations, interactive visualizations, images, and more. Jupyter was spawned from its parent project, IPython, in 2014 as usage of the Notebook grew outwards from its origins in scientific computing and the Python programming language to the emerging worlds of data science, machine learning, and a host of other programming languages such as Julia and R. Organizationally, Jupyter is community governed and fiscally sponsored by the non-profit NumFOCUS foundation[2].

Since the release of the Notebook in 2011, Jupyter has created a number of other open-source subprojects that address other aspects of this space: 1) JupyterLab[3], the project's next-generation, extensible notebook user interface, 2) nbconvert[4], for converting notebooks to other formats, 3) Jupyter Widgets[5], for building interactive graphical interfaces in notebooks, 4) Voilà[6], for turning notebooks into dashboards and web applications, 5) JupyterHub[7], for multiuser deployments of Jupyter, 6) Binder[8], a service for turning Git repositories into live Jupyter servers for ad-hoc exploration of notebook-based content, and 7) nbviewer[9], a service for previewing notebooks hosted online, and a number of others.

[1] https://jupyter.org
[2] https://numfocus.org
[3] https://github.com/jupyterlab/jupyterlab
[4] https://github.com/jupyter/nbconvert
[5] https://github.com/jupyter-widgets/ipywidgets
[6] https://github.com/voila-dashboards/voila
[7] https://github.com/jupyterhub/jupyterhub
[8] https://mybinder.org
[9] https://nbviewer.jupyter.org

Today, Jupyter Notebooks have become ubiquitous across computational education and research, science, data science, and machine learning. Many millions of users and tens of thousands of organizations use Jupyter on a daily basis. As of early 2021, there are over 10 million public Jupyter Notebooks on GitHub alone[10]. Major research collaborations and communities across physics, chemistry, biology, economics, Earth science, etc. leverage Jupyter as a foundational tool for their computational work, collaboration, education, and knowledge dissemination. Entire curricula at universities and massive open online courses (MOOCs) are based on Jupyter. All major cloud providers and multiple startups offer products and services based on Jupyter notebooks.

Given the scope of Jupyter's usage and the constraints of this article, it is impossible to do justice to all of the remarkable things that users are doing with Jupyter. Instead we refer the reader to the talks from JupyterCon 2020[11], along with the papers in this issue of CISE for further exploration.

Similarly, this article cannot do justice to everything that the large, diverse, and widespread community of Jupyter contributors has built, both on the software and community sides. Everything we describe here rests on a 20-year open collaboration where stakeholders from academia, industry, government, and more have participated as peers. More information about the Jupyter Distinguished Contributors and Steering Council can be found in the project's website[12].

As co-founders and co-directors of Jupyter, the two of us have been asked to introduce Jupyter to readers of this CISE issue, which represent a small fraction of the content from JupyterCon 2020. At the same time, 2021 marks the 10th anniversary of the Jupyter Notebook and the 20th anniversary of IPython. As such, we believe it is worth pausing and asking two questions. **What are the main ideas of Jupyter and why have Jupyter Notebooks turned out to be so useful to such a wide range of users and domains?**

The overarching idea of Jupyter is that humans matter. The context of this statement is that in data science and computationally intensive research and development, the weight of technical concerns often dominates: algorithms, programming languages, systems and software architecture, and so on. In this context, human concerns and problems are often secondary at best. Jupyter lives in this universe: its software and users are technically sophisticated and its primary usage case is solving complex problems with code and data. In spite of this, we claim that the primary problems that Jupyter solves are uniquely human. What are these human problems that Jupyter solves? To answer this, we briefly discuss three dimensions of Jupyter: interactive computing, computational narratives, and the idea that Jupyter is more than software. We conclude by describing how these ideas have enabled communities of practice to be created across a broad range of computational domains.

Interactive computing

At the most basic level, Jupyter provides an architecture and applications for interactive computing. We claim that interactive computing solves a human problem: it enables humans to leverage computers and data to perform a broad spectrum of human tasks: decide, analyze, understand, accept, reject, discover, question, predict, create, hypothesize, test, evaluate, and play. Or more simply, Jupyter helps humans to think. This is seen in a human-centered definition of interactive computing.

For our purposes, **an interactive computation is a persistent computer program that runs with a "human in the loop," where the primary mode of interaction is through the same human iteratively writing/running blocks of code and looking at the results.**

First, these programs are persistent and stateful: the program has working memory which records the results of previous computations, and which are available in subsequent computations. Second, the user

[10] https://github.com/parente/nbestimate
[11] https://www.youtube.com/c/JupyterCon/videos
[12] https://jupyter.org/about

provides input to the program by writing code instead of using graphical, touch or other interfaces. Third, in contrast to graphical user interfaces (GUIs) or most of software engineering, in this flavor of interactive computing, a single human is both the user and author of the program. Fourth, in contrast to software engineering, there is no externally specified goal or design target. Instead, the user explores and discovers their goal as they gain understanding from iteratively executing the code and thinking about the results and their data.

This definition of interactive computing is rooted in the modern scientific computing community. Tools such as IDL (1977), Maple (1982), Matlab (1984), and Mathematica (1988) offer this mode of interaction. It shouldn't be surprising that the two of us grew up as physicists using these interactive computing tools as a foundational part of our computational workflows. Indeed, we created IPython and Jupyter initially because we wanted the same type of interactive computing experience in the Python programming language.

We acknowledge that our definition of interactive computing here is somewhat narrow. The entire field of human-computer interaction (HCI) is concerned with how humans interact with computers across all modes of interaction. Of the many ways to interact with computers, writing code is perhaps the most inhumane (imagine if we had to write code to post to Twitter or send emails…). Why is writing code so effective for some tasks and activities?

From a computer-centered perspective, interactive computing has been formalized by modelling these systems as persistent Turing machines coupled to an environment (the human user) [2]. More colloquially, the simplest expression of an interactive computation is the REPL, or read-eval-print loop. In a REPL, the program repeatedly reads lines of code, evaluates that code, and then prints the result. Simple terminal-based interactive shells like IPython, as well as the Jupyter Notebook, follow this pattern with minor variations. But the computer-centered perspective doesn't answer the question posed above: what is the value of a REPL from the human perspective?

To answer this, let's flip the REPL around and cast it from the perspective of the human user. The user has a counterpart to the computer's read-eval-print-loop: a "write-eval-think-loop" (WETL). The user first **writes** a block of code to import data, train a model, create a visualization, implement an algorithm, etc. (which the computer then reads). The user then asks the computer to **evaluate** that block of code (which the computer does). And then, after the computer displays the result, the user looks at that result and **thinks** about what to do next. Is this what I expected to see? How is X related to Y? Why was an exception raised? What might I predict using this dataset and what features would be useful? In short, the user is thinking with code and data.

As this iteration proceeds, the human user and computer work together and converse through code and its output. Because the language of this conversation is a programming language like Python, the user is able to think about complex technical problems, algorithms, and data. This idea of a computer being used as a thinking companion is not new:

> [Human]-computer symbiosis is an expected development in cooperative interaction between [humans] and electronic computers… to enable [humans] and computers to cooperate in making decisions and controlling complex situations without inflexible dependence on predetermined programs. In the anticipated symbiotic partnership, [humans] will set the goals, formulate the hypotheses, determine the criteria, and perform the evaluations. Computing machines will do the routinizable work that must be done to prepare the way for insights and decisions in technical and scientific thinking [3].

Ultimately, understanding, and the responsibility of making decisions based on this understanding, are fundamentally human activities. As a tool for interactive computing, Jupyter enables users to apply computation and data to challenging questions in contexts as diverse as climate change, policy, public health, research, business operations, justice, legislation, and more.

Computational narratives

While IPython and other REPLs/WETLs offer an interactive computing experience that enables users to think with code and data, they lack permanence. More specifically, these tools help a user to think in the moment, but when a session is closed there are no persistent artifacts that can be used to share, disseminate, or reproduce the work. This is the second human problem that Jupyter solves and brings us to the idea of a computational narrative.

Narrative is universal. Humans are evolved to create, share, and consume narratives or stories [4]; [5]. All known cultures practice storytelling and regardless of culture or education, humans acquire the ability and inclination to create and process stories at a young age [4]. Much of our waking hours are spent producing and consuming narratives. Indeed, it is difficult to have a conversation without telling a story:

> Storytelling and understanding are functionally the same thing... intelligence is bound up with our ability to tell the right story at the right time [6].

This narrative-centered aspect of human understanding stands in contrast to computers, which are optimized to consume, produce, and process data. In order for data and the computations that process and visualize that data to be useful to humans, they must be embedded into a narrative—a computational narrative—that tells a story for a particular audience and context[13].

Computational notebooks were introduced by Mathematica in 1988; the Jupyter Notebook is our concrete realization of the computational narrative, with a web-based architecture designed for extensibility, programming language independence, and an open document format. The raw events of this narrative are the input (code) and output of the iterations of the REPL/WETL of the underlying interactive computation. Around that, the user can add narrative text including equations, multi-media content, and more.

[13] See H. Porter Abbott's *Cambridge Introduction to Narrative* (2008) for an excellent overview of narrative that covers the topic in a manner that applies naturally to computational narratives.

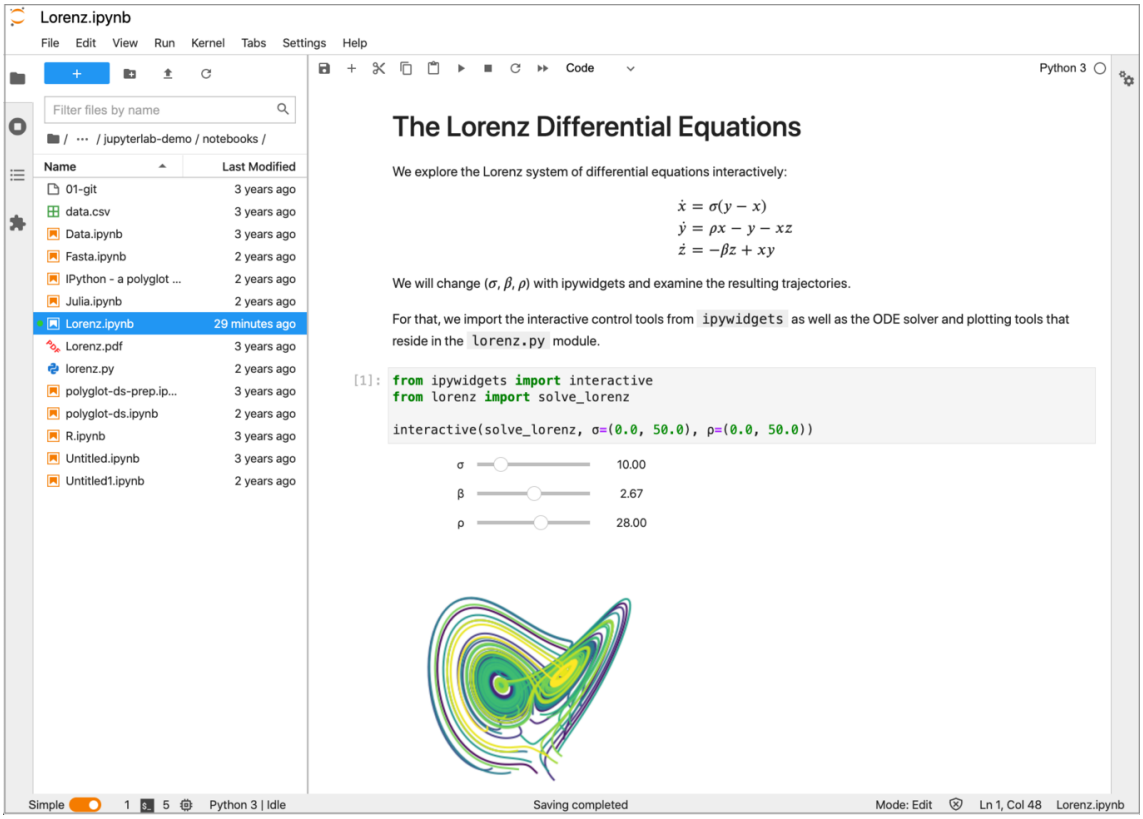

Figure 1: An example Jupyter notebook that solves the Lorenz differential equations [7], open in JupyterLab. This notebook has live interactive code, headings, narrative text, equations, visualizations, and interactive controls that are organized into a human centered computational narrative.

Computational narratives built on the Jupyter Notebook solve a number of human problems. First, they make interactive computations reproducible as a natural byproduct of work. Second, they provide an artifact that can be shared with others, version-controlled, used for communicating results, etc. Third, because Jupyter notebooks use an open format, they can be converted into other forms, including websites, books, online documentation, dashboards.

These human uses of computational narratives illustrate how and why the ways that Jupyter notebooks are used are so dramatically different from traditional software engineering tools where the goal is for one group of people to write software that is subsequently deployed to and used by an entirely different group of users. While Knuth's literate programming paradigm [8] weaves human-oriented documentation into the software engineering process, a computational narrative is distinct in its incorporation of interactive computing as the central element. The outcome here is not a software product but ideas and understanding that are "deployed" to other humans.

More than software

While Jupyter's open source software is obviously central to the project, over the years we have developed a broader perspective that guides the project and has been a primary factor in its growth and adoption: Jupyter is more than merely software. More specifically, Jupyter also builds and consists of services, open standards and protocols, and community.

For many users, content in their domain is the first reason to learn about Jupyter: services like nbviewer and Binder allow them to read, share and execute computational narratives about topics of relevance to them, from blog posts to research papers and interactive textbooks. The Jupyter software and services directly support these learning and knowledge sharing goals.

Next, in addition to building software, Project Jupyter has developed open standards and protocols for interactive computing that our software implements. This has allowed third parties to build an ecosystem of interoperable software without explicitly sharing code. The JSON-based Jupyter Notebook document format[14] enables first- and third-party tools to use and combine Jupyter Notebooks for a wide range of purposes such as: i) third-party user interfaces for working with Jupyter Notebooks (Colab[15], nteract[16], CoCalc[17], VSCode[18]), and ii) tools for converting notebooks and displaying them online as standalone websites, books, etc. or within other applications (nbviewer, Jupyter Book[19], GitHub[20], Authorea[21]). Jupyter has also developed a network protocol to communicate between interactive computing user interfaces (the Jupyter Notebook, JupyterLab, terminal-based REPLs, etc.) and the server-side computational processes that run users' code (called kernels in Jupyter's architecture). This kernel message protocol[22] has enabled third parties to: i) adapt Jupyter's runtime architecture to novel deployment scenarios, ii) build over 100 different kernels supporting most programming languages in use today[23], and iii) build alternate interactive computing applications that reuse Jupyter's runtime architecture.

Furthermore, Jupyter's software offers multiple programmatic APIs, which facilitate customization and extension without forking or copying the entire code base. For example, the Jupyter server usually stores users' files on a local filesystem. However, the server offers an API that third parties have leveraged to store notebooks on Amazon S3, relational databases, and Google Drive. Users can install and use any combination of these extensions. The architecture of Jupyter's next-generation user interface, JupyterLab, also illustrates this pattern: the entire application is a set of extensions that are standalone JavaScript packages. These can be composed into new tools that meet the needs of the users, without the project having to implement every conceivable feature.

This brings us to community. In addition to being software, Jupyter is a community of users, developers and stakeholders from all walks of life. While IPython and Jupyter were initially built by a small team of developers, today over 1,500 people have contributed to our codebase and many more build content anchored in our ecosystem. Furthermore, hundreds of third-party developers participate in this community and build extensions, applications, and content that leverage Jupyter. The pinnacle of the community are Jupyter users, who number in the millions. This community is not accidental: the core Jupyter team has invested significant effort into welcoming new contributors, helping users, planning and running community events (Jupyter Community Workshops[24], JupyterDays and JupyterCon[25]), and training and mentoring junior developers and designers. Critically, we have developed, and continue to refine, an open governance model that seeks to meet the needs of such diverse stakeholders, whose combined effort has enabled the impact of Jupyter to grow in an organic manner that far exceeded the resources of the core contributors.

These additional dimensions of Jupyter beyond software are visualized in Figure 2.

[14] https://pypi.org/project/nbformat
[15] https://colab.research.google.com
[16] https://github.com/nteract/nteract
[17] https://cocalc.com
[18] https://code.visualstudio.com
[19] https://jupyterbook.org
[20] https://github.com
[21] https://www.authorea.com
[22] https://github.com/jupyter/jupyter_client
[23] https://github.com/jupyter/jupyter/wiki/Jupyter-kernels
[24] https://blog.jupyter.org/jupyter-community-workshops-call-for-proposals-for-jan-aug-2020-710f687e30f4
[25] https://jupytercon.com

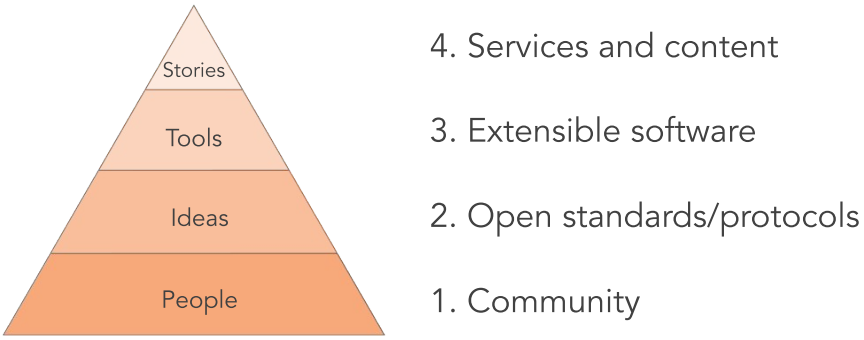

Figure 2: Jupyter has a layered ecosystem that 1) is founded on a diverse community of users and contributors who 2) establish open standards and protocols, by 3) building extensible software, that is 4) deployed in services and enables the authoring and sharing of content. These layers build on each other, are each irreplaceable, and drive innovation.

Communities of practice (CoP)

Communities of Practice (CoP) are groups of people motivated by a common set of problems or topics, who collectively develop accepted practices, often aimed at the advancement of knowledge in a specific professional domain [9]. A central element of CoP is therefore the ability to fluidly share knowledge, and in particular, to share it in ways that enable others to reuse the shared work and build upon it. Jupyter has become an enabling technology for CoP that operate in technical spaces where computing, data analysis, and programming are central.

Several elements in the Jupyter ecosystem play complementary roles in support of CoP. Most prominently, the computational narratives of Jupyter notebooks support both individual exploration of ideas and sharing of the resulting knowledge in a reusable, reproducible manner that encourages feedback and collaboration. In turn, a body of knowledge encoded in such narratives fuels the cycle of collaboration that builds the CoP. These narratives are particularly valuable in research and education, fields where exploration, discovery, reproducibility, and shared understanding of complex problems are key objectives.

Furthermore, other aspects of Jupyter beyond Notebooks support the growth of CoP, as we illustrate now.

Notebook sharing: when the IPython Notebook was released in 2011, sharing work in notebooks would require the recipient to also have the software installed to view it, or to ask the author to convert the notebook to a widely used format like HTML or PDF. The nbviewer service made this conversion a one-click action. Originally prototyped by M. Bussonnier in 2012, it enabled anyone to easily share the rendered HTML version of any publicly available notebook as a link that readers could access with a web browser. We observed a rapid rise of notebook sharing via blogs and social media, as people would publish their work in this format. This pattern of sharing has continued and expanded as other platforms, such as GitHub, have added builtin notebook rendering. Today, Jupyter Book facilitates sharing entire collections of notebooks that form complete interactive "textbooks". These can be hosted online as static HTML websites at no cost via tools like GitHub Pages, and as live, executable notebooks via Binder.

Group usage: while the Notebook was designed as a single-user application running on a personal com-

puter, basing it on web technologies made it possible to host it on a remote or cloud-based server while providing an identical user experience. JupyterHub, first released in 2015 by Min Ragan-Kelley and other members of the team, offered this capability: it could host Notebooks on a remote server for multiple concurrent users, with authenticated access. This made it possible for groups to work on shared infrastructure. University courses were an obvious and early use case: we both teach courses in data science at our respective universities, hosted entirely on cloud-based JupyterHubs. This pattern has been adopted by many colleges and universities and has even reached K-12 education with efforts like the Callysto project in Canada[26]. Research groups and industry teams similarly adopted JupyterHub as a tool to build shared computational infrastructure. Today, scientific HPC facilities including NERSC, NCAR, and Canada's Syzygy[27] offer national-level infrastructure accessible to scientists through the web browser thanks to HPC-hosted JupyterHubs.

Reproducible sharing: while nbviewer allows the sharing of static narratives, with the release of Binder, originally prototyped by Jeremy Freeman and Andrew Osheroff in late 2015, it became possible to share a live, executable version of one or more notebooks with similar ease. Binder turns a repository of notebooks with explicitly declared dependencies into a live, ephemeral container in the cloud that can be accessed with a web browser and where the user can immediately execute all the code for free without having to download or install any of the underlying software.

These examples from the Jupyter architecture and ecosystem illustrate how open, modular tools amplify the value of individual notebooks and support the sharing of knowledge in ways that facilitate CoP. We have seen CoP that use Jupyter heavily in fields as diverse as bioinformatics, high-energy physics, data science, machine learning, music, economics and more. A compelling example is the geoscience and climate CoP built around the Pangeo project[28], "A community platform for Big Data geoscience," which we discuss in more detail now.

Pangeo, an NSF-funded project from the EarthCube program, defines itself as "first and foremost a community of people working collaboratively to develop software and infrastructure to enable Big Data geoscience research." The Pangeo team identified the following problems limiting progress in modern geoscience and climate research: fluid access to large-scale datasets, lack of technological sophistication in the tools available to scientists, and reproducibility. Originally led by Ryan Abernathey and Joe Hamman, the Pangeo team identified that open-source tools already met these challenges fairly well, though with a "last-mile problem" of configuration, deployment, and documentation for the specific needs of the Earth and climate science community.

Pangeo adopted JupyterHub, configured with Xarray for access to numerical datasets, and Dask for distributed computing, as the backbone of their platform. The open, vendor-agnostic nature of the Jupyter tools made it possible for Pangeo to be deployed in the cloud or on HPC hardware, thus bridging the traditional practices of scientific computing with today's cloud-hosted tools and datasets. They have deployed custom tools like a Dask plugin that provides real-time feedback of distributed processing in JupyterLab, and specialized Binders with Dask support that help this CoP meet the challenge of reproducibility in large-scale workflows.

Pangeo's adoption of Jupyter has enabled it to grow and develop in a number of directions where Jupyter plays a key role:

- HackWeeks [10] are events that combine education in computational methods with community building and research prototyping around domain-specific topics. A number of HackWeeks have been hosted on Pangeo Hubs, including satellite data analysis for cryosphere science using NASA's ICESat-2, oceanography, broad-spectrum geosciences, and large-scale climate modeling with the CMIP6 data.

[26] https://www.callysto.ca
[27] https://syzygy.ca
[28] https://pangeo.io

- The Pangeo Gallery[29] offers live collections of notebooks hosted on Dask-enabled Binder deployments. Topics covered range from technical infrastructure (e.g., using Dask for scalable computing, illustrations of the Pangeo software stack, or performance benchmarks for cloud-based data analysis) to domain science (e.g., analysis of Landsat8 imagery, processing of remote sensing and simulation-based ocean data, study of the National Weather Model, modeling of water levels under hurricanes, reproducing key papers in global climate modeling, and more).
- The *Jupyter meets the Earth*[30] project connects developments in Jupyter with research use cases in the geosciences, including some from the Pangeo-supported HackWeeks on ICESat-2 and CMIP6.
- *Project Pythia*[31] develops new learning materials for the analysis of geoscience data using the Scientific Python ecosystem. These last two projects are co-led by members of Pangeo and funded under the same NSF EarthCube umbrella.

The Pangeo community has built successful demonstrations of how to conduct large-scale science in the cloud, and this need exists in areas beyond geoscience. Other domain communities are leveraging this approach for their own needs, such as the PanNeuro effort led by Ariel Rokem and others at the University of Washington, and we see this as a positive sign that Jupyter's infrastructure has broad reach for different communities.

The Jupyter team has always sought to create tools that are not only open, but vendor-agnostic, modular and extensible. While the Pangeo CoP has their own specific needs and requirements, the same building blocks can be customized, extended, and deployed by other CoP and service providers to other usage cases. The key point here is how the the open-source building blocks of Jupyter for interactive computing and computational narratives unlock new practices and collaboration patterns in these communities.

In closing

This takes us back to the main idea of this article, which we believe summarizes the past, present, and future direction of Project Jupyter. Jupyter helps individuals and groups to leverage computation and data to solve complex, technical, but human-centered problems of understanding, decision making, collaboration, and community practice. Furthermore, we believe that open-source, community governed, modular and extensible software that is built using the principles and pratices of human-centered design are particularly effective in tackling these challenges.

Acknowledgments

We thank Lorena Barba and Hans Fangohr for their editorial work on this special issue and for helpful comments on the manuscript. We thank Lindsey Heagy for helpful feedback and work on Figure 2. Most importantly, we thank all IPython/Jupyter contributors, whose work over two decades has made this project possible, as well as the broader Scientific Python community that Jupyter relies on. B.G and F.P. acknowledge funding for Jupyter from the Alfred P. Sloan Foundation, the Gordon and Betty Moore Foundation, the Helmsley Charitable Trust, and Schmidt Futures. F.P. acknowledges current funding by the NSF EarthCube program under awards 1928406, 1928374. B.G. aknowledges Amazon Web Services for time to contribute to Jupyter and this article.

Brian Granger is a Principal Technical Program Manager at Amazon Web Services in the AI Platform team and spent the last decade as a professor of physics and data science at Cal Poly State University in San Luis

[29] http://gallery.pangeo.io
[30] https://bit.ly/jupytearth
[31] https://ncar.github.io/ProjectPythia

Obispo, CA. His work focuses on building open-source tools for interactive computing, data science, and data visualization. Brian is a co-founder of Project Jupyter, co-founder of the Altair project for statistical visualization, and an active contributor to a number of other open-source projects focused on data science in Python. He is an advisory board member of NumFOCUS and a faculty fellow of the Cal Poly Center for Innovation and Entrepreneurship. Along with other leaders of Project Jupyter, he is a winner of the 2017 ACM Software System Award. Brian has a Ph.D. in theoretical atomic, molecular, and optical physics from the University of Colorado, Boulder. Contact him at bgranger@calpoly.edu

Fernando Pérez is an associate professor in Statistics at UC Berkeley and scientist at LBNL. He builds open-source tools for humans to use computers as companions in thinking and collaboration, mostly in the scientific Python ecosystem (IPython, Jupyter & friends). His current research focuses on questions in geoscience and how to build the computational and data ecosystem to tackle problems like climate change with collaborative, open, reproducible, and extensible scientific practices. He is a co-founder of the 2i2c.org initiative, the Berkeley Institute for Data Science and the NumFOCUS Foundation. He is a National Academy of Science Kavli Frontiers of Science Fellow and a member of the Python Software Foundation. He received the 2017 ACM Software System Award and the 2012 FSF Award for the Advancement of Free Software. Pérez holds a PhD in physics from the University of Colorado, Boulder. Contact him at fernando.perez@berkeley.edu

References

[1]T. Kluyver *et al.*, "Jupyter Notebooks – a publishing format for reproducible computational workflows", in *Positioning and Power in Academic Publishing: Players, Agents and Agendas*, 2016.

[2]D. Goldin, S. A. Smolka, and P. Wegner, *Interactive Computation*. Springer Berlin Heidelberg, 2006.

[3]J. C. R. Licklider, "Man-Computer Symbiosis", *IRE Transactions on Human Factors in Electronics*, vol. HFE-1, no. 1, pp. 4–11, Mar. 1960, doi: 10.1109/thfe2.1960.4503259.

[4]M. S. Sugiyama, "Narrative Theory and Function: Why Evolution Matters", *Philosophy and Literature*, vol. 25, no. 2, pp. 233–250, 2001, doi: 10.1353/phl.2001.0035.

[5]K. Young and J. L. Saver, "The Neurology of Narrative", *SubStance*, vol. 30, no. 1/2, p. 72, 2001, doi: 10.2307/3685505.

[6]R. Schank, *Tell Me a Story: Narrative and Intelligence (Rethinking Theory)*. Northwestern University Press, 1995.

[7]E. N. Lorenz, "Deterministic nonperiodic flow", *Journal of Atmospheric Science*, vol. 20, 1963.

[8]D. E. Knuth, "Literate Programming", *The Computer Journal*, vol. 27, no. 2, pp. 97–111, Feb. 1984, doi: 10.1093/comjnl/27.2.97.

[9]E. Wenger, "Communities of Practice: Learning as a Social System", *Systems Thinker*, 1998.

[10]D. Huppenkothen, A. Arendt, D. W. Hogg, K. Ram, J. T. VanderPlas, and A. Rokem, "Hack weeks as a model for data science education and collaboration", *Proceedings of the National Academy of Sciences*, vol. 115, no. 36, pp. 8872–8877, Aug. 2018, doi: 10.1073/pnas.1717196115.